CS 134: Networks Problem set 2

Prof. Yaron Singer Due: **Wednesday 2/8/2017**

Resources: lecture notes 3 and 4 (on Canvas), Kleinberg and Easley's *Networks, Crowds, and Markets* Chapter 2; Watts and Strogatz, Collective Dynamics of Small Worlds; Kleinberg, The Small World Phenomenon: An Algorithmic Perspective; section notes week 2 (on Canvas).

1. The square lattice. (20 points) In the following question we will consider a k-square lattice on n nodes. Recall that a 1-square lattice on n nodes with $r = \sqrt{n}$ can be thought of as a graph in which each node has some unique index (i, j) where $i, j \in \{1, \ldots, r\}$ and is connected to nodes with indices $(i-1, j), (i+1, j), (i, j-1), (i, j+1)$[1]. The *lattice distance* between two nodes u and v is the number of edges between them on the 1-square lattice. A k-square lattice is a square lattice with the additional edges connecting every two nodes whose lattice distance is at most k. That is, a node u whose index on the lattice is (i, j) is connected to all nodes v whose indices are (x, y) where $|i - x| + |j - y| \leq k$. Consider such a k-square lattice on n nodes:

a. (6 points) Compute its diameter for $k = 2$.

b. (8 points) Consider a node u that is at lattice distance at least 2 from any nodes on the edge of the lattice (i.e. the node u has index (i, j) where $3 \leq i, j \leq r - 2$). Compute the clustering coefficient of node u (as defined in the previous problem set) for $k = 2$.

c. (6 points) In the case of $k = 1$, and for even r, show that the expansion parameter of this graph is smaller than $2/r$. In other words, show that there exists a set of nodes S of size smaller or equal to $n/2$ (or $r^2/2$) has at most $|S| \cdot 2/r$ edges leaving it.

Solution: Thanks, Richard Ouyang!

[1] More precisely, (i, j) is connected to nodes $(i-1, j)$ if $i > 1$, $(i+1, j)$ if $i < r$, $(i, j-1)$ if $j > 1$, and $(i, j+1)$ $j < r$.

1 The square lattice

(a) The farthest distance between any two vertices in a 1-square lattice with $n = r \times r$ nodes is $2(r-1)$, since going from one corner to the opposite corner takes $2(r-1)$ steps. Then, we can halve the number of steps by turning the graph into a 2-square lattice, since we can jump a lattice distance of 2 now. Thus the diameter of such a graph is $r - 1 = \sqrt{n} - 1$.

(b) Consider a 5×5 graph, and note that we're looking for the clustering coefficient of the middle node (which has index (3, 3)). This node has 12 edges (up, down, left, right, two up, two down, two left, two right, and the diagonals), so the denominator of the clustering coefficient is $\binom{12}{2} = 66$. Now count the edges for each of these 12 nodes. The nodes (1, 3), (3, 1), (3, 5), (5, 3) each have 3 neighbors that are also neighbors of (3, 3). The nodes (2, 3), (3, 2), (3, 4), (4, 3) each have 6 such neighbors, and the remaining nodes (2, 2), (2, 4), (4, 2), (4, 4) also have 6 such neighbors, for a total of 60 edges. But we have overcounted by a factor of 2, since we're counting both sides of an edge. So the final clustering coefficient is $\frac{30}{66} = \frac{5}{11}$.

(c) To prove this, we need only to show one such set S. Consider an $r \times r$ square lattice (where r is even), and let S be the left $r \times \frac{r}{2}$ of the lattice. Then it's clear that there are $r = \frac{r^2}{2}\frac{2}{r} = |S| \cdot \frac{2}{r}$ edges leaving the graph, and the expansion of this graph is at most $\frac{2}{r}$. For odd r (for example $r = 3$), this isn't true.

2. A smaller world? (Easley and Kleinberg, 20.8 Q1) (20 points) In the basic six-degrees-of-separation question, one asks whether most pairs of people in the world are connected by a path of at most six edges in the social network, where an edge joins any two people who know each other on a first-name basis. Now let's consider a variation on this question. Suppose that we consider the full population of the world, and suppose that from each person in the world we create an edge only to their ten closest friends (but not to anyone else they know on a first-name basis)[2]. In the resulting "closest-friend" version of the social network, is it possible that for each pair of people in the world, there is a path of at most six edges connecting this pair of people? Explain.

Solution: Thanks, Matthew Huang! Take any arbitrary person v. He must be connected to the entire world (which is $\leq 10^9$ people) by 6 or fewer edges. We simply put an upper bound on the number of people P he can be connected to via 6 edges. Let $p(k)$ be the set of people he is connected to with a distance of exactly k. Then,

$$P \leq |p(1)| + |p(2)| + \cdots + |p(6)|$$

Note that $|p(1)| = 10$. For $|p(2)|$, an upper bound is realized when each member of $p(1)$ is connected to 9 people (1 is reserved for connecting back to v), all non-overlapping with other neighbors of members of $p(1)$. Thus, $|p(2)| \leq 10 \cdot 9 \leq 10^2$. We can repeat this process for all $p(k)$, so that $|p(k)| \leq 10^k$. This means:

$$P \leq 10 + 10^2 + \cdots + 10^6 \leq 10^9$$

Thus, if even one person cannot be connected to the entire world, then the small world theory does not hold under this social network.

[2]Assume that the relationship "closest friend" is symmetric, i.e. if Alice considers Bob to be one of her 10 closest friends, than Bob also considers Alice to be one of his 10 closest friends.

3. Keep your friends close, and your acquaintances closer. (Easley and Kleinberg, 20.8 Q2) (20 points) We consider another variation of the six-degrees-of-separation question. For each person in the world, we ask them to rank the thirty people they know best, in descending order of how well they know them. (Let's suppose for purposes of this question that each person is able to think of thirty people to list.) We then construct two different social networks:

- The "close-friend" network: from each person we create a directed edge only to their ten closest friends on the list.
- The "distant-friend" network: from each person we create a directed edge only to the ten people listed in positions 21 through 30 on their list.

Let's think about how the small-world phenomenon might differ in these two networks. In particular, let C be the average number of people that a person can reach in six steps in the close-friend network, and let D be the average number of people that a person can reach in six steps in the distant-friend network (taking the average over all people in the world). When researchers have done empirical studies to compare these two types of networks (and the exact details often differ from one study to another), they tend to find that one of C or D is consistently larger than the other.

a. (8 points) For which of the close-friend network and the distant-friend network do you expect the clustering coefficient to be the largest?

b. (8 points) In which of the close-friend network and the distant-friend network do you expect to see the "long-range" edges that we talked about in the Watts-Strogatz and the Kleinberg models?

c. (4 points) Which of the two quantities, C or D, do you expect to be larger?

Give a brief explanation for each answer.

Solution: Thanks, Javier Cuan-Martinez!

3. (a) I expect that the **close-friend network** would have the largest clustering coefficient. Logistically, the closest people that anyone person will know will also know each other because of the average frequency that the person in question sees all of them. For example, the closest people that a person will know will probably be a mother, father, and siblings, all of which know each other. Perhaps the closest people that they know are on the swim team, all of which know each other.

   (b) I expect to see the long-range edges in the **distant-friends network** because the purpose of the long-range edges in the models that we studied is to simulate the random process by which people can randomly meet each other. These random individuals can be very distant from the person in question, so there is going to be less of a chance of clustering occurring. In the close-friends model, there will be less long-range edges occurring because there is a higher chance that everyone knows each other, thereby, classifying the created edge as not long-range.

   (c) I expect the D to be larger. This is because the neighbors of the root person in this model will have friends that the root person wouldn't normally meet. This would allow for longer jumps along the graph of connections. The close-friend model has too much clustering, which wouldn't allow edges to travel too far because each neighbor that is visited would still be really close to the root person.

4. Regular graphs. (20 points) In this question you need to construct a graph that has certain properties. You can describe the graphs as a list, matrix, or drawing. We consider a graph with n nodes, and you can either be creative in describing a graph that generalizes to n nodes or simply use $n = 12$.

a. **(6 points)** Describe a connected 2-regular graph with diameter $n/2$ and clustering coefficient 0.

b. **(14 points)** Describe a connected 4-regular graph with diameter $n/4$ and clustering coefficient 1/2.

Solution: Thanks, Tomislav Zabcic-Matic!

(a) Here, we are dealing with a simple ring graph with $k = 1$ of arbitrary size $n > 3$. In this graph, all nodes have degree 2, making it 2-regular, and since no pairs of outgoing edges from any node are connected to form a triangle, the clustering coefficient of the graph is 0.

(b) In this case, we are dealing with a ring lattice of size n such that 4 divides n. This ring lattice has $k = 2$. This makes it possible to travel any distance on the graph in $\frac{n}{4}$ steps. For the clustering coefficient, it is enough to look at one node and generalize:
Each node has 4 outgoing edges, which means it has 6 pairs of edges. the two edges in the standard ring are connected by a "$k = 2$" edge. The two pairs of adjacent ring and "$k = 2$" edges are connected by other ring edges. The two pairs of opposite ring and "$k = 2$" edges are not connected, and finally, the pair of "$k = 2$" edges is not connected. Thus, for each node, we have clustering coefficient $\frac{3}{6} = \frac{1}{2}$. This generalizes to the entire graph, since all vertices have the same form.

5. Coding: short distances in networks. (20 points) Compute distances on networks

- (7 points) Recall the Watts-Strogatz model discussed in class. In this model, n nodes are positioned on a square lattice. Each node is connected to the nodes that are at most a distance k away on the lattice for some given parameter $k \in \mathbb{N}$. Moreover, each node has l *long range* edges which are connected uniformly at random to other nodes in the network.
For $r = 1, 2, \ldots, 10$ (or $n = 1, 4, \ldots, 100$), generate networks according to the Watts-Strogatz model with $k = 2$ and $l = 2$. Plot the average distance between any two nodes as a function of n and plot the function $\log n$;

- (6 points) In the Kleinberg model, like in the Watts-Strogatz model, n nodes are positioned on a square lattice, and each node is connected to the nodes that are at most a distance k away on the lattice for some given parameter $k \in \mathbb{N}$. Now, for some given l, each node has l long range edges that are generated as follows:

$$\Pr[u \to v] = \frac{\Delta(u,v)^{-2}}{\sum\limits_{w \neq u} \Delta(u,w)^{-2}}$$

$\Pr[u \to v]$ is the probability that the long range edge is created linking nodes u and v and $\Delta(u, v)$ is the lattice distance between nodes u and v.
For $n = 1, \ldots, 100$, generate networks according to the Kleinberg model with $k = 2$ and $l = 2$. Plot the average distance between any two nodes as a function of n against $\log^2 n$;

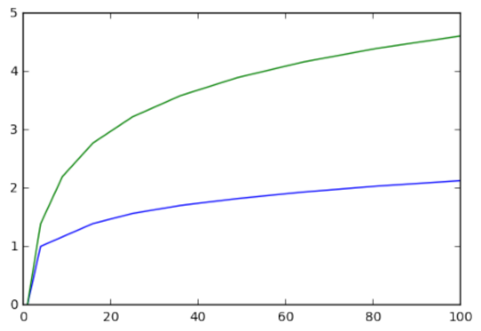
- (7 points) Compute a **good estimate** (note: this does not need to be exact; clever solutions that provide good approximations of the shortest path are preferable to brute-force solutions) of the average shortest distances of these graphs:

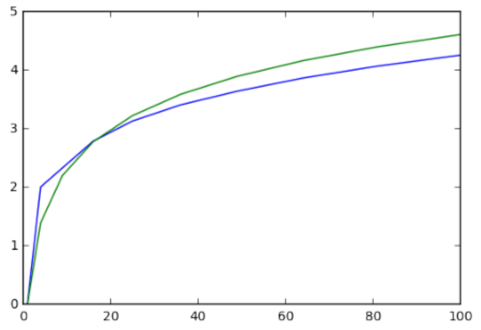
  `http://www.hcs.harvard.edu/~cs134-spring2017/wp-content/uploads/2017/01/enron.txt`
  `http://www.hcs.harvard.edu/~cs134-spring2017/wp-content/uploads/2017/01/epinions.txt`
  `https://www.dropbox.com/s/rzi61yf9f8rzgdv/livejournal.txt.zip?dl=0`

  These files are formatted as lines of tab-separated values. For each line, $n_1 \quad n_2$ corresponds to an edge from node n_1 to node n_2. In your solution, please include 1) your approximations of the average shortest distances of these graphs, 2) a brief written explanation describing your approach, 3) submit your code on Canvas. We have no specific style guidelines for your code, but please make it as easily-readable as you can.

Solution: Thanks Matt Huang!

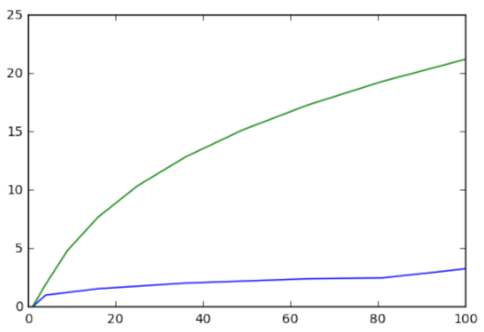
We implement Watts-Strogatz, and plot the average distance (blue) versus $\log n$ (green). If we scale the average distance by a constant factor (second graph), the two graphs align:

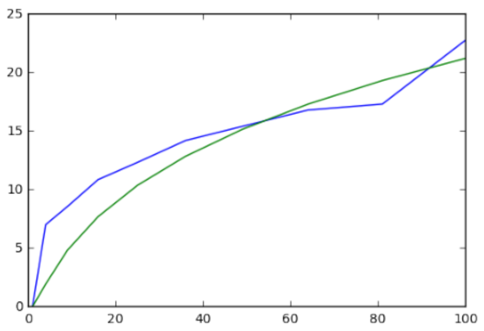




This is reassuring—the small world model stipulates that people are separated by a distance on the order of $\log n$, and this is consistent with Watts-Strogatz being a strong small-world model.

We implement Kleinberg, and plot the average distance (blue) versus $\log n$ (green). If we scale the average distance by a constant factor (second graph), it is plausible that the two graphs are asymptotically comparable.





Again, this is reassuring since the average distance is poly-log, consistent with the small-world model.

Our approximations for the average shortest path are:

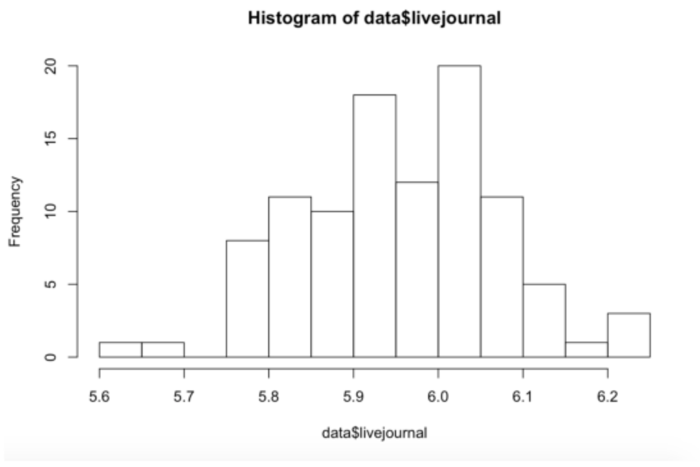
`enron`	`epinions`	`livejournal`
4.041 ± 0.102	4.753 ± 0.150	5.960 ± 0.118

For each file, we inputted the graph data into an adjacency list. We then select a random node u, and perform a single-source BFS to compute the shortest distance between u and every node in the graph. This process gives us a "one-source average distance" that we compute. We compute 30 one-source average distances by selecting random notes, and we average them to obtain one estimation of the average pairwise distance.

We take 100 estimations (a total of 3000 one-source average distances), and take the mean and standard deviation of these values. Since each estimation consists of 30 trials, the distribution of estimations is approximately normal, by the CLT. Therefore, the standard deviation of our 100 estimations is a more appropriate report of our confidence (since one-source average distances can vary widely in the same graph). The histogram of estimations below shows that our estimations we tend to a normal distribution.

This algorithm tends to the correct average distance, since we are repeating 3000 one-source average distances. Each one-source average distance is the average of $\{\Delta(u, v_1), \Delta(u, v_2), \ldots, \Delta(u, v_{n-1})\}$. With 3000 trials, we are averaging $3000(n-1)$ pairwise distances, each sampled with equal probability. Therefore, our randomized algorithm computes an accurate average of the average pairwise distance.

Code was written in C++ and is attached in the appendix. Runtime was approximately 10 hours in total.


Histogram of data$livejournal
Frequency
0
5
10
15
20
5.6
5.7
5.8
5.9
6.0
6.1
6.2
data$livejournal

Appendix for Code

Watts-Strogatz and Kleinberg

```
#include <queue>
#include <vector>
#include <random>
using namespace std;
typedef pair<int, int> pii;
typedef vector< vector<int> > vvi;

int r,n;
const int k = 2;
const int l = 2;
const int T = 100; //trials
const int R = 10; //max value of r

inline int dist(int x1, int y1, int x2, int y2) {return abs(x1-x2) + abs(y1-y2);}

// encoded index -> (x,y) grid coordinates
void decode(int ind, int& x, int& y) {
    x = ind/r;
    y = ind % r;
}

//distance between two encoded indices
int ind_dist(int ind1, int ind2) {
    int x1,x2,y1,y2;
    decode(ind1, x1, y1);
    decode(ind2, x2, y2);
    return dist(x1, y1, x2, y2);
}

void gen_klattice(vvi& G) {
    //make G[i][i] = 1, we won't use these values anyways
    for(int i = 0; i < n; ++i)
        for(int j = 0; j < n; ++j)
            if(ind_dist(i,j) <= k)
                G[i][j] = 1;
}

//long range edges for watts-strogatz
void gen_ws_lre(vvi& G) {
    for(int i = 0; i < n; ++i)
        for(int c = 0; c < l; ++c) {
            int rn;
            while((rn = rand() % n) == i);
            G[i][rn] = 1;
            G[rn][i] = 1;
        }
}
inline float inv_sq(int x) { return 1/(float(x) * float(x)); }

//long range edges for kleinberg
void gen_kleinberg_lre(vvi& G) {
    for(int i = 0; i < n; ++i) {
        vector<float> pdensity(n, 0);
        for(int j = 0; j < n; ++j)
            if(j != i)
                pdensity[j] = inv_sq(ind_dist(i, j));

        default_random_engine generator(time(NULL));
        discrete_distribution<int> dd(pdensity.begin(), pdensity.end());

        for(int c = 0; c < l; ++c) {
            int ind = dd(generator);
            G[i][ind] = 1;
        }
    }
}

//single-source bfs
void update_dists(vvi& G, int st, vvi& D) {
    //pairs (dist, node)
    queue<pii> Q;
    vector<bool> vis (n, false);
    Q.push(make_pair(0, st));
    while(!Q.empty()) {
        pii p = Q.front();
        Q.pop();
        int d = p.first;
        int node = p.second;
        if(vis[node])
            continue;

        D[st][node] = d;
        D[node][st] = d;
        vis[node] = true;
        for(int i = 0; i < n; ++i)
            if(G[node][i] && !vis[i])
                Q.push(make_pair(d + 1, i));
    }
}

//naive all-pairs distances
void get_pair_dists(vvi& G, vvi& D) {
    for(int i = 0; i < n; ++i)
        update_dists(G, i, D);
}




//update running sum
void update_stats(vvi& G, vvi& D, int& dsum, int& cnt) {
    for(int i = 0; i < n; ++i)
        for(int j = 0; j < n; ++j) {
            if(i == j)
                continue;
            dsum += D[i][j];
            cnt++;
        }
}
```

```
void watts_strogatz() {
    printf("WATTS-STROGATZ\n");
    for(r = 2; r <= R; ++r) {
        int dist_sum = 0;
        int pair_cnt = 0;
        n = r * r;
        for(int t = 0; t < T; ++t) {
            vvi G (n, vector<int> (n, 0));
            gen_klattice(G);
            gen_ws_lre(G);

            vvi pair_dists (n, vector<int> (n, 0));
            get_pair_dists(G, pair_dists);

            update_stats(G, pair_dists, dist_sum, pair_cnt);
        }
        float f = float(dist_sum) / pair_cnt;
        printf("r = %d: average distance %.5f\n", r, f);
    }
}

void kleinberg() {
    printf("KLEINBERG\n");
    for(r = 2; r <= R; ++r) {
        int dist_sum = 0;
        int pair_cnt = 0;
        n = r * r;
        for(int t = 0; t < T; ++t) {
            vvi G (n, vector<int> (n, 0));
            gen_klattice(G);
            gen_kleinberg_lre(G);

            vvi pair_dists (n, vector<int> (n, 0));
            get_pair_dists(G, pair_dists);

            update_stats(G, pair_dists, dist_sum, pair_cnt);
        }
        float f = float(dist_sum) / pair_cnt;
        printf("r = %d: average distance %.5f\n", r, f);
    }
}
int main(int argc, char** argv) {
    srand(time(NULL));
    watts_strogatz();
    kleinberg();
}
```

Average Distance Approximation

```
#include <iostream>
#include <queue>
#include <vector>
#define Rd(r) freopen(r, "r", stdin)
using namespace std;
typedef long long ll;
typedef vector< vector<int> > vvi;
typedef pair<int, int> pii;

const int MAXN = 5000000;
const int k = 30; //to be approximately normally distributed
const int T = 100;
int N = 0;

//single-source bfs
void bfs(vvi& G, int st, double& dist_sum, double& count_pairs) {
    //pairs (dist, node)
    queue<pii> Q;
    vector<bool> vis (N, false);
    Q.push(make_pair(0, st));
    while(!Q.empty()) {
        pii p = Q.front();
        Q.pop();
        int d = p.first;
        int curr = p.second;
        if(vis[curr])
            continue;

        dist_sum += double(d);
        count_pairs++;

        vis[curr] = true;

        for(int i = 0; i < G[curr].size(); ++i) {
            int next = G[curr][i];
            if(!vis[next])
                Q.push(make_pair(d + 1, next));
        }
    }
    //account for 0-distance self-"edge"
    count_pairs--;
}

int main(int argc, char** argv) {
    if(argc < 2) {
        cerr << "Usage: " << argv[0] << " <datafile>" << endl;
        return 1;
    }
    if(!Rd(argv[1])) {
        cerr << "Error opening file " << argv[1] << endl;
        return 1;
    }

    srand(time(NULL));
    vector< vector<int> > G(MAXN, vector<int>());
    int n1, n2;
    while(cin >> n1 >> n2) {
        N = max(n1, N);
        G[n1].push_back(n2);
    }

    cerr << "Done reading input" << endl;
    for(int t = 0; t < T; ++t) {
        double dist_sum = 0;
        double count_pairs = 0;
        for(int i = 0; i < k; ++i) {
            int rn = rand() % N;
            bfs(G, rn, dist_sum, count_pairs);
        }
        double avg_dist = dist_sum/count_pairs;
        cout << avg_dist << endl;
    }
}
```